Improving Reliability of Cloud-Based Applications

Hong Thai Tran, George Feuerlicht

Improving Reliability of Cloud-based Applications

Hong Thai Tran[1] and George Feuerlicht [1, 2, 3]

[1] Faculty of Engineering and Information Technology, University of Technology, Sydney, hongthai.tran@uts.edu.au, george.feuerlicht@uts.edu.au
[2] Unicorn College, V Kapslovně 2767/2,130 00 Prague 3, Czech Republic,
[3] Department of Information Technology, University of Economics, Prague, W. Churchill Sq. 4, Prague 3, Czech Republic

Abstract. With the increasing availability of various types of cloud services many organizations are becoming reliant on providers of cloud services to maintain the operation of their enterprise applications. Different types of reliability strategies designed to improve the availability of cloud services have been proposed and implemented. In this paper we have estimated the theoretical improvements in service availability that can be achieved using the Retry Fault Tolerance, Recovery Block Fault Tolerance and Dynamic Sequential Fault Tolerance strategies, and we have compared these estimates to experimentally obtained results. The experimental results obtained using our prototype Service Consumer Framework are consistent with the theoretical predictions, and indicate significant improvements in service availability when compared to invoking cloud services directly.

Keywords: Reliability of Cloud Services, Fault Tolerance, RFT, RBFT, DSFT

1 Introduction

With the increasing use of cloud services the reliability of enterprise applications is becoming dependent on the reliability of consumed cloud services. In the public cloud context, service consumers do not have control over externally provided cloud services and therefore cannot guarantee the levels of security and availability that they are typically expected to provide to their users [1]. While most cloud service providers make considerable efforts to ensure the reliability of their services, cloud service consumers cannot assume continuous availability of cloud services, and are ultimately responsible for the reliable operation of their enterprise applications. In response to such concerns, hybrid cloud solutions have become popular [2]; according to Gartner Special Report on the Outlook for Cloud [3] half of large enterprises will adopt and use the hybrid cloud model by the end of 2017. Hybrid cloud solutions involve on-premise enterprise applications that utilize external cloud services, for example Paypal Payment Gateway (www.paypal.com), cloud storage service Amazon S3 (aws.amazon.com/s3), or entire SaaS (Software as a Service) applications. With a hybrid delivery model where enterprise applications are partially hosted on premise and partially in the cloud, enterprises can balance the benefits and drawbacks of both

approaches, and decide which applications can be migrated to the cloud and which should be deployed locally to ensure high levels of data security and privacy. However, from the reliability point of view, hybrid cloud introduces a number of significant challenges as IT (Information Technology) infrastructure and enterprise applications become fragmented over multiple environments with different reliability characteristics.

Another reliability challenge concerns service evolution, i.e. changes in functional attributes of services that may impact on existing consumer applications. Services are often the subject of uncontrolled changes as service providers implement functional enhancements and rectify defects with service consumers unable to predict when or how services will change [4]. Consequently, service consumers suffer service disruptions and are forced to upgrade their applications to maintain compatibility with new versions of cloud services, often without any notification. As the complexity of service-oriented applications grows, it is becoming imperative to develop effective methods to manage service evolution and to ensure that service consumers are protected from service changes.

In this paper we describe the reliability features of the Service Consumer Framework (SCF) designed to improve the reliability of cloud-based enterprise applications by managing service outages and service evolution. In the next section (section 2) we review related literature dealing with the reliability of cloud-based solutions. In section 3 we describe three reliability strategies (Retry Fault Tolerance, Recovery Block Fault Tolerance, and Dynamic Sequential Fault Tolerance) and calculate their expected theoretical impact on the probability of failure and response time. In section 4 we discuss how these reliability strategies are implemented using the SCF framework. Section 5 describes our experimental setup and gives a comparison of the theoretical results calculated in section 3 with the experimental measurements of availability and response time. Section 6 contains our conclusions and proposals for future work.

2 Related work

Traditional approaches to developing reliable, fault tolerant on-premise SOA (Service Oriented Architecture) applications include fault prevention and forecasting. For example, Tsai, Zhou [5] propose a SOA testing and evaluation framework that implements group testing to enhance test efficiency. This framework uses coverage relationships and recent test results to rank and to eliminate test cases with overlapping coverage. Using redundancy-based fault tolerance strategies, Zibin and Lyu [1] propose a distributed replication strategy evaluation and selection framework for fault tolerant web services. Authors compare various replication strategies and propose a replication strategy selection algorithm. Developing highly reliable cloud-based applications introduces a number of new reliability challenges, as enterprise applications are no longer under the full of control of local developers and administrators. In response to such challenges, Zibin, Zhou [6] present a FTCloud component ranking framework for building fault-tolerant cloud applications. Using two different ranking algorithms: structure-based component ranking and hybrid component ranking, au-

thors identify the most critical components of cloud applications and then determine an optimal fault-tolerance strategy for these components. Based on this work, Reddy and Nalini [7] propose FT2R2Cloud as a fault tolerant solution using time-out and retransmission of requests for cloud applications. FT2R2Cloud measures the reliability of the software components in terms of the number of responses and the throughput. Authors propose an algorithm to rank software components based on their reliability calculated using a number of service outages and service invocation.

In recent research, Zhengping, Nailu [2] propose the S5 system accounting framework to maximize reliability of cloud services. The framework consists five different layers: service existence examination, service availability examination, service capability and usability examination, service self-healing layer, and system accounting user interface. Authors also propose a new definition of quality of reliability for cloud services. In another work, Adams, Bearly [8] describe fundamental reliability concepts and a reliability design-time process for organizations. Authors provide a guideline for IT architects to improve the reliability of their services and propose processes that architects can use to design cloud services that mitigate potential failures. More recently, Zheng and Lyu [9] identified major problems when developing fault tolerance strategies and introduced the design of static and dynamic fault tolerance strategies. Authors identify significant components of complex service-oriented systems, and investigate algorithms for optimal fault tolerance strategy selection. A heuristic algorithm is proposed to efficiently solve the problem of selection of a fault tolerance strategy. The authors describe an algorithm for component ranking aiming to provide a practical fault-tolerant framework for improving the reliability of enterprise applications. Zheng, Lyu [10] describe a Retry Fault Tolerance (RFT) that involves repeated service invocations with a specified delay interval until the service invocation succeeds. This strategy is particularly useful in situations characterized by short-term outages.

Focusing on improving the reliability of cloud computing, Chen, Jin [11] present a lightweight software fault-tolerance system called SHelp, which can effectively recover programs from different types of software faults. SHelp extends ASSURE [12] to improve its effectiveness and efficiency in cloud computing environments. Zhang, Zheng [13] propose a novel approach called Byzantine Fault Tolerant Cloud (BFTCloud) to manage different types of failures in voluntary resource clouds. BFTCloud deploys replication techniques to overcome failures using a broad pool of nodes available in the cloud. Moghtadaeipour and Tavoli [14] propose a new approach to improve load balancing and fault tolerance using work-load distribution and virtual priority.

Another aspect of cloud computing that impacts on reliability involves service evolution. Service evolution has been the subject of recent research interest [4, 15-19], however, the focus of these activities so far has been mainly on developing methodologies that help service providers to manage service versions and deliver reliable services. In the cloud computing environments where services are provided externally by independent organizations (cloud service providers) a consumer-side solution is needed to ensure the reliability of cloud-based service-oriented application [10].

3 Reliability Strategies

In this section we discuss three reliability strategies that are implemented in the SCF framework: Retry Fault Tolerance (RFT), Recovery Block Fault Tolerance (RBFT), and Dynamic Sequential Fault Tolerance (DFST). As noted above in section 2, these strategies have been described in the literature for on-premise systems [6], [7], [10]. We have adapted the RFT, RBFT and DFST reliability strategies for cloud services to address short-term and long-term service outages, and issues arising from service evolution. Short-term outages are situations where services become temporarily inaccessible, for example as a result of the loss of network connectivity; automatic recovery typically restores the service following a short delay. Long-term service outages are typically caused by scheduled and unscheduled maintenance or system crashes that require service provider intervention to recover the service. Service evolution involves changes in functional characteristics of services associated with functionality enhancements and changes aimed at improving service performance. Service evolution may involve changes to service interfaces, service endpoints, security policy, or may involve service retirement. Most cloud service providers maintain multiple versions of services to limit the impact of such changes on service consumers, and attempt to ensure backward compatibility between service versions. However, in practice it is not always possible to avoid *breaking* consumer applications, resulting in a situation where service consumers are forced to modify their applications to ensure compatibility with the new version of the service. Service overload occurs when the number of service requests in a given time period exceeds the provider limit.

3.1 Retry Fault Tolerance

Retry Fault Tolerance (Figure 1) is a relatively simple strategy commonly used in enterprise application. Using this strategy, cloud services are repeatedly invoked following a delay period until the service invocation succeeds. RFT helps to improve reliability, in particular in situations characterized by short-term outages. The overall probability of failure (PF_{RFT}) can be calculated by:

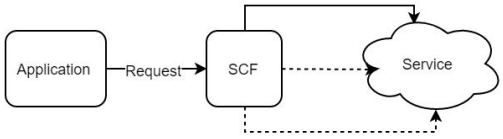

Fig. 1. Retry Fault Tolerance

$$PF_{RFT} = PF^m \tag{1}$$

where PF is the probability of failure of the service and m is a number of retry attempts. While RFT reduces the probability of failure, it may increase the overall response time T_{RFT} due to delays between consecutive service invocations. The total delay can be estimated as:

$$T_{RFT} = \sum_{i=1}^{m}(T_{(i)} + D \times (i-1)) \times (PF)^{i-1} \tag{2}$$

where D is the delay between retry attempts and T_i is the response time of i[th] invocation. The above calculations assume independent modes of failure of subsequent invocations; this assumption only holds in situations where the delay D is much greater than the duration of the outage, i.e. for long duration outages the invocation will fail repeatedly, invalidating the assumption of independence of failures of subsequent invocations.

3.2 Recovery Block Fault Tolerance

Recovery Block Fault Tolerance (Figure 2) is a widely used strategy that relies on service substitution using alternative services invoked in a specified sequence. It is used to improve the availability of critical applications. The failover configuration includes a *primary* cloud service used as a default (active) service, and *stand-by* services that are deployed in the event of the failure of the primary service, or when the primary service becomes unavailable because of scheduled/unscheduled maintenance. Now assuming independent modes of failure, the overall probability of failure for *n* services combined can be computed by:

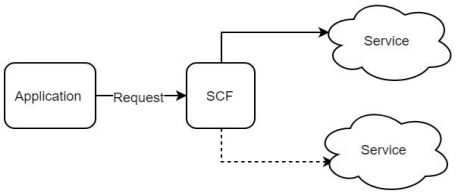

Fig. 2. Recovery Block Fault Tolerance

$$PF_{RBFT} = \prod_{i=1}^{n} PF_{(i)}\,;\; A_{RBFT} = 1 - PF_{RBFT} \tag{3}$$

where n is the total number of services and PF_i is the probability of failure of the i[th] alternative service. The overall response time $T(s)$ can be calculated by:

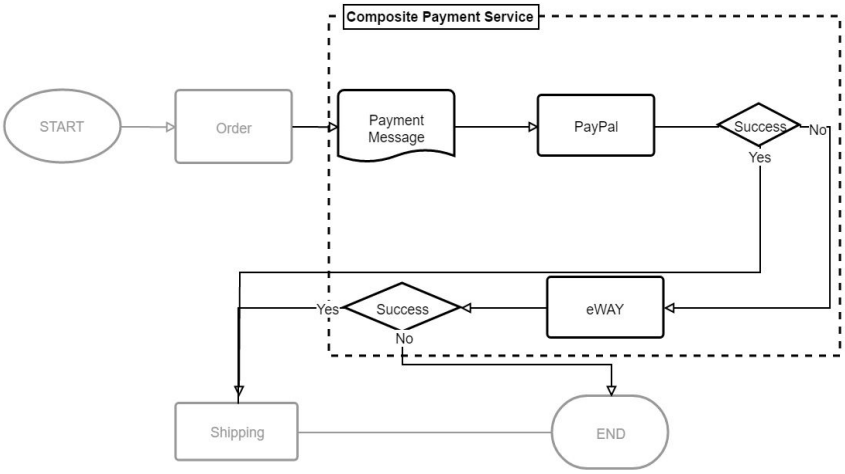

Fig. 3. Online shopping scenario using a composite payment service

$$T_{RBFT} = T_{(1)} + \sum_{i=2}^{n}(T_{(i)} \times \prod_{k=1}^{i-1} PF_{(k)}) \quad (4)$$

where T_1 is response time of first service invocation and T_i is response time of i[th] alternative service invocation. In the online shopping scenario illustrated in Figure 3, the composite payment service uses eWay payment service as an alternative (stand-by) service for the PayPal (primary) service. Assuming that the availability of both PayPal and eWay services is 99.9% (corresponding to an outage of approximately 9 hour per year), and that the probability of failure PF = 0.01 for each service, the overall RBFT probability of failure PF = 10^{-6}, and the overall availability A_{RBFT} s = 99.9999% (this corresponds to an outage of approximately 5 minutes per year).

3.3 Dynamic Sequential Fault Tolerance

The Dynamic Sequential Strategy (Figure 4) is a combination of the RFT and RBFT strategies. When the primary service fails following RFT retries, the dynamic sequential strategy will deploy an alternative service. The overall probability of failure for the *n* services combined is given by:

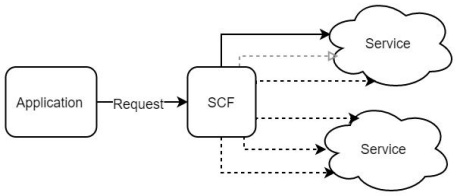

Fig. 4. Dynamic Sequential Strategy

$$PF_{DSFT} = \prod_{i=1}^{n} PF_{RFT(i)}\,;\; A_{DSFT} = 1 - PF_{DSRF} \quad (5)$$

where $PF_{RFT(i)}$ is the probability of failure of the i[th] alternative service using the RFT strategy calculated in equation (1), and $A(s)$ is the overall availability of the composite service using the DSFT strategy. The overall response time $T(s)$ can be calculated by:

$$T_{DSFT} = \mathrm{T}_{RFT(1)} + \sum_{i=2}^{n}(\mathrm{T}_{RFT(i)} \times \prod_{k=1}^{i-1} \mathrm{PF}_{RFT(k)}) \quad (6)$$

where $\mathrm{T}_{RFT(1)}$ is the response time of the first service using the RFT strategy in equation (2), $\mathrm{T}_{RFT(i)}$ is response time of i[th] alternative service calculated in equation (2), and $\mathrm{PF}_{RFT(k)}$ is the probability of failure of the k[th] alternative service using the RFT strategy calculated using equation (1). Table 1 indicates the suitability of the RFT, RBFT, and DSFT strategies to different types of reliability challenges.

Table 1. Suitability of reliability strategies

Method	Short Outages	Long Outages	Service Evolution	Service Overload
RFT	yes	no	no	no
RBFT	yes	yes	yes	yes
DSFT	yes	yes	yes	yes

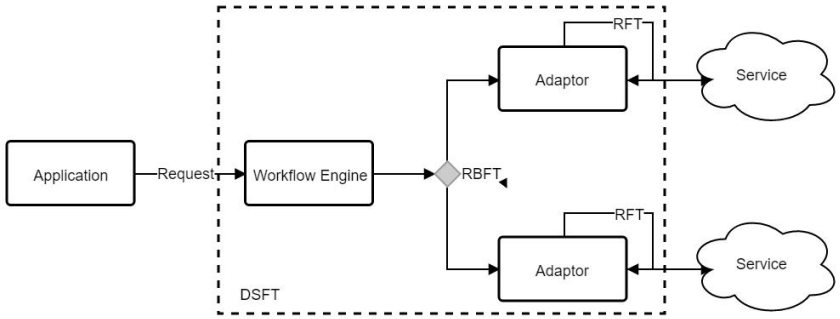

Fig. 5. Service Consumer Framework reliability features

4 Implementation of Reliability Strategies using the SCF

The SCF framework is designed to manage hybrid cloud environments and aims to address the main issues that impact on the reliability of enterprise applications. The SCF framework implements RFT, RBFT and DSFT strategies and is briefly described in the following sections. The framework consists of four main components: Service Repository, Workflow Engine, Service Adaptors and a Notification Centre. Detail description of the SCF framework can be found in [19]. Figure 5 illustrates how service adaptors and the workflow engine can be configured to implement the various reliability strategies.

4.1 Service repository

Service repository maintains information about the available services and adaptors, including metadata that describes functional and non-functional attributes of certified services. The information held in the service repository is used to manage services and to design reliable applications. The functional and non-functional QoS (Quality of Service) attributes held in the service repository enable the selection of suitable services by querying the service repository specifying desired service attributes. Services with identical (or similar) functionality are identified to indicate that these services can be used as alternatives to the primary service to implement the RBFT strategy.

4.2 Service adaptors

Service adaptors are *connectors* that integrate software services with enterprise applications. Each cloud service recorded in the repository is associated with a corresponding service adaptor. Service adaptors use a native interface to transform service requests to a request that is compatible with the current version of the corresponding cloud service, maintaining compatibility between enterprise applications and external services. The function of a service adaptor is to invoke a service, keep track of service

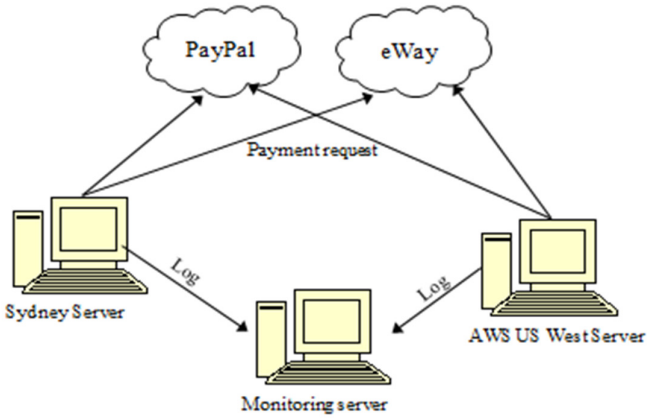

Fig. 6. Experimental Configuration

status, and record service execution information in the service repository. Service adaptors implement the RFT reliability strategy by configuring the number of retry attempts and the delay period.

4.3 Workflow engine

The workflow engine implements service workflows, facilitates service failover and the composition of services. The workflow engine executes workflows and routes requests to corresponding cloud services. Workflows can be configured to implement the RBFT strategy by using a number of alternative services redundantly. Another important function of the workflow engine is load balancing. Service adaptors can be configured as *active* or *stand by*. By default, active service adaptors are used to process the requests and stand by adaptors are deployed in situation when the primary (active) adaptor requests fail or when the primary adaptor becomes overloaded.

4.4 Notification Centre

The SCF framework maintains execution logs and updates service status records in the service repository. When service faults occur, notification centre notifies application administrators so that a recovery action can take place. The administrators are able to rapidly react to service failures and maintain application availability minimizing downtime. In addition, execution logs are used to monitor services and to analyze QoS attributes.

5 Experimental Verification of Reliability Strategies

Figure 6 illustrates the experimental configuration that was used to verify the theoretical calculations in section 3. The experimental setup consists of two servers that

host the SCF framework and a separate Monitoring Server. Both SCF servers implement identical payment scenario illustrated in Figure 3 using PayPal Pilot service (pilot-payflowpro.paypal.com) and eWay Sandbox (https://api.sandbox.ewaypayments.com). The payment requests are randomly generated and sent to the PayPal and eWay payment servers from two different locations. The US West Server uses Amazon Web Services (AWS) cloud-based infrastructure located on the West Coast of the United States and has a high quality server with a reliable network connection. The Sydney server is a local server in Sydney, Australia with a less reliable Internet connection.

5.1 Experimental Setup

We have collected experimental results from both servers for a period of thirty days, storing the data in the logs on the Monitoring Server deployed on AWS. The log data records were analyzed computing the experimental values of availability and response time for the composite payment service using different reliability strategies. Both SCF servers generate payment requests in intervals varying randomly between 5 and 10 seconds, and use the following four strategies:

Strategy 1: Payment requests are sent directly to the payment service without applying any reliability strategy

Strategy 2: Payment requests are sent to the payment service using RFT strategy with three retry attempts (R=3) and a delay of five seconds (D=5)

Table 2. Consumer Service Transaction Logs

Service	Start Time	Response Time	Result
eWay	31/03/2016 12:51	3.59 seconds	Success
PayPal	31/03/2016 12:50	1.48 seconds	Success
PayPal	31/03/2016 12:50	1.39 seconds	Success
PayPal	31/03/2016 12:49	1.39 seconds	Success
eWay	31/03/2016 12:49	2.50 seconds	Success
PayPal	31/03/2016 12:48	1.44 seconds	Success
eWay	31/03/2016 12:48	1.51 seconds	Success
eWay	31/03/2016 12:47	1.72 seconds	Success
PayPal	31/03/2016 12:47	1.41 seconds	Success
PayPal	31/03/2016 12:46	1.39 seconds	Success
eWay	31/03/2016 12:46	2.17 seconds	Success
PayPal	31/03/2016 12:45	1.39 seconds	Success
PayPal	31/03/2016 12:45	1.00 seconds	Error
eWay	31/03/2016 12:44	2.14 seconds	Success

Strategy 3: Payment requests are sent to a composite payment service using the RBFT strategy

Strategy 4: Payment requests are sent to a composite payment service using the DSFT strategy which is combination of RBFT and RFT with PayPal (R=3, D=5) and eWay (R=3, D=5)

5.2 Experimental Results

We have collected the payment transaction data independently of the values available from the cloud service providers, storing this information in the log files on the Monitoring Server (Table 2 shows a fragment of the response time measurements). The use of two separate servers in two different locations enables the comparison of availability and response time information collected under different connection conditions.

Table 3. Availability of payment services

Server	PayPal	eWay	PayPal RFT	eWay RFT	PayPal-eWay RBFT	PayPal-eWay DSFT
US West	90.48%	93.86%	97.90%	97.26%	99.80%	99.95%
Sydney	90.18%	93.53%	97.28%	97.03%	99.29%	99.82%

As shown in Table 3, using Strategy 1 (i.e. without deploying any reliability strategy) the availability for PayPal and eWay services on the US West server is 90.4815% and 93.8654%, respectively. Deploying the RFT strategy (Strategy 2) the availability increases to 97.9033% and 97.2607%, for PayPal and eWay services, respectively. Using the RBFT strategy (Strategy 3), the availability of the composite service (PayPal and eWay) increases to 99.8091%, and finally using the DSFT strategy (Strategy 4) the availability of the composite service (PayPal + eWay) increases further to 99.9508%. The theoretical values obtained in section 3 are slightly higher than the experimental values; this can be explained by noting that connection issues may affect both PayPal and eWay services concurrently, invalidating the assumption of independent modes of failure.

Table 4. Response time of payment services in seconds

Server	PayPal	eWay	PayPal RFT	eWay RFT	PayPal-eWay RBFT	PayPal-eWay DSFT
US West	1.35	1.84	2.33	2.66	2.06	2.44
Sydney	3.70	1.75	5.67	3.02	4.98	5.43

Table 4 shows the average response time of PayPal and eWay services using different reliability strategies during the period from March 15th to April 15th 2016. The average response time of the US West Server is considerably lower than the Sydney Server when connecting to the PayPal service in the US. However, for the eWay service (https://www.eway.com.au/), which is located in Australia, the response time of

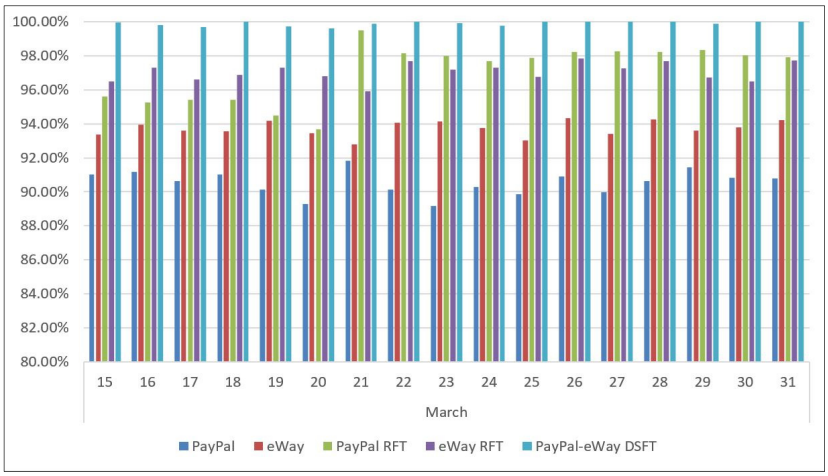

Fig. 7. Availability of reliability strategies from 15th to 31st of March

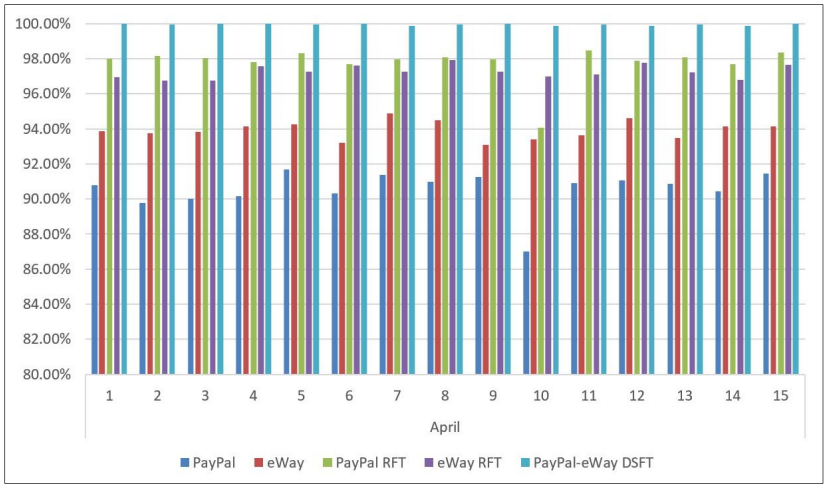

Fig. 8. Availability of reliability strategies from 1st to 15th of April 2016

the Sydney Server is slightly better than for the US West server. The bar charts in Figure 7 and 8 give a comparison of the availability values for various reliability strategies for the period of March 15th to April 15th 2016. As the figures illustrate, the availability of PayPal and eWay services using any of the reliability strategies is significantly higher than without deploying a reliability strategy. During the measurement period the availability of the PayPal service varied between 88% and 92%, but the availability of the combined PayPal-eWay services using the DSFT strategy remained above 99.9%.

6 CONCLUSIONS

With the increasing availability of various types of cloud services many organizations are becoming reliant on providers of cloud services to maintain the operation of their enterprise applications. Different types of strategies designed to improve the availability of cloud services have been proposed and implemented. These reliability strategies can be used to improve availability of cloud-based enterprise applications by addressing service outages, service evolution, and failures arising from overloaded services.

In this paper we have estimated the theoretical improvements in service availability that can be achieved using the Retry Fault Tolerance, Recovery Block Fault Tolerance, and Dynamic Sequential Fault Tolerance strategies and compared these values to experimentally obtained results. The experimental results obtained using the SCF framework are consistent with theoretical predictions, and indicate significant improvements in service availability when compared to invoking cloud services directly (i.e. without deploying any reliability strategy). In the specific case of payment services, the availability for PayPal and eWay services increased from 90.4815% and 93.8654%, respectively for direct payment service invocation, to 97.9033% and 97.2607%, for PayPal and eWay services, respectively when the RFT strategy was used. Using the RBFT strategy, the availability of the composite service (PayPal + eWay) increased to 99.8091%, and using the DSFT strategy the availability of the composite service (PayPal + eWay) increased further to 99.9508%.

Deploying multiple alternative services using the RBFT strategy also alleviates issues arising from service evolution that results in incompatible versions of services released by service providers. We are currently extending the functionality of the SFC framework to detect such situations and automatically redirect requests to alternative services.

References

1. Zibin, Z. and M.R. Lyu. A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services. in Web Services, 2008. ICWS '08. IEEE International Conference on. 2008.
2. Zhengping, W., C. Nailu, and S. Peng. Improving Cloud Service Reliability -- A System Accounting Approach. in Services Computing (SCC), 2012 IEEE Ninth International Conference on. 2012.
3. Rivera, J. and R.v.d. Meulen, Gartner Says Nearly Half of Large Enterprises Will Have Hybrid Cloud Deployments by the End of 2017, in Gartner Special Report Examines the Outlook for Hybrid Cloud2013.
4. Andrikopoulos, V., S. Benbernou, and M.P. Papazoglou, On the Evolution of Services. IEEE Transactions on Software Engineering, 2012. 38(3): p. 609-628.
5. Tsai, W.T., et al., On Testing and Evaluating Service-Oriented Software. Computer, 2008. 41(8): p. 40-46.
6. Zibin, Z., et al., Component Ranking for Fault-Tolerant Cloud Applications. Services Computing, IEEE Transactions on, 2012. 5(4): p. 540-550.

7. Reddy, C.M. and N. Nalini. FT2R2Cloud: Fault tolerance using time-out and retransmission of requests for cloud applications. in Advances in Electronics, Computers and Communications (ICAECC), 2014 International Conference on. 2014.
8. Adams, M., et al. An introduction to designing reliable cloud services. Microsoft Trustworthy Computing 2014; Available from: https://www.microsoft.com/en-au/download/details.aspx?id=34683.
9. Zheng, Z. and M.R. Lyu, Selecting an Optimal Fault Tolerance Strategy for Reliable Service-Oriented Systems with Local and Global Constraints. IEEE Transactions on Computers, 2015. 64(1): p. 219-232.
10. Zheng, Z., M. Lyu, and H. Wang, Service fault tolerance for highly reliable service-oriented systems: an overview. Science China Information Sciences, 2015. 58(5): p. 1-12.
11. Chen, G., et al., A lightweight software fault-tolerance system in the cloud environment. Concurrency and Computation: Practice and Experience, 2015. 27(12): p. 2982-2998.
12. Sidiroglou, S., et al., ASSURE: automatic software self-healing using rescue points, in Proceedings of the 14th international conference on Architectural support for programming languages and operating systems2009, ACM: Washington, DC, USA. p. 37-48.
13. Zhang, Y., Z. Zheng, and M.R. Lyu. BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing. in Cloud Computing (CLOUD), 2011 IEEE International Conference on. 2011.
14. Moghtadaeipour, A. and R. Tavoli. A new approach to improve load balancing for increasing fault tolerance and decreasing energy consumption in cloud computing. in 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI). 2015.
15. Eisfeld, A., D.A. McMeekin, and A.P. Karduck. Complex environment evolution: Challenges with semantic service infrastructures. in DEST-2012. 6th IEEE International Conference on Digital Ecosystems Technologies. 2012.
16. Romano, D. and M. Pinzger. Analyzing the Evolution of Web Services Using Fine-Grained Changes. in ICWS-2012. IEEE 19th International Conference on Web Services. 2012.
17. Zhenmei, Y. and L. Fengming. Small-World Based Trust Evaluation Model for Web Service. in Computer Science & Service System (CSSS), 2012 International Conference on. 2012.
18. Ziyan, X., Z. Haihong, and L. Lin, User's Requirements Driven Services Adaptation and Evolution, in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual2012. p. 13-19.
19. Feuerlicht, G. and H.T. Tran. Service consumer framework: Managing Service Evolution from a Consumer Perspective. in ICEIS-2014. 16th International Conference on Enterprise Information Systems. 2014. Portugal: Springer.